**Language and Computers**  
**Writers' Aids**  

Based on Dickinson, Brew, & Meurers (2013)  
Indiana University  
Fall 2013

---

**Why people care about spelling**

- Misspellings can cause misunderstandings.
- Standard spelling makes it easy to organize words & text:
  - e.g., Without standard spelling, how would you look up things in a lexicon or thesaurus?
  - e.g., Optical character recognition software (OCR) can use knowledge about standard spelling to recognize scanned words even for hardly legible input.
- Standard spelling makes it possible to provide a single text, accessible to a wide range of readers (different backgrounds, speaking different dialects, etc.).
- Using standard spelling can make a good impression in social interaction.

---

**How are spell checkers used?**

- **Interactive spelling checkers** = spell checker detects errors as you type.
  - It may or may not make suggestions for correction.
  - It needs a “real-time” response (i.e., must be fast)
  - It is up to the human to decide if the spell checker is right or wrong, and so we may not require 100% accuracy (especially with a list of choices)
- **Automatic spelling correctors** = spell checker runs on a whole document, finds errors, and corrects them
  - A much more difficult task.
  - A human may or may not proofread the results later.

---

**Detection vs. Correction**

- There are two distinct tasks:
  - **Error detection** = simply find the misspelled words
  - **Error correction** = correct the misspelled words
  - e.g., It might be easy to tell that *ater* is a misspelled word, but what is the correct word? *water*? *later*? *after*?
  - Note that detection is a prerequisite for correction.

---

**Error causes**

**Keyboard mistypings**

- **Space bar issues**
  - **run-on** errors = two separate words become one
    - e.g., *the fuzz becomes the fuzz*
  - **split errors** = one word becomes two separate items
    - e.g., *equalization becomes equali zation*
  - Note that the resulting items might still be words!
    - e.g., *a tollway becomes a toll way*

- **Keyboard proximity**
  - e.g., *Jack becomes Hack since h and j are next to each other on a typical American keyboard*

- **Physical similarity**
  - similarity of shape, e.g., *mistaking two physically similar letters when typing up something handwritten*
    - e.g., *light for fight*
Error causes

**Phonetic errors**
- errors based on the sounds of a language (not necessarily on the letters)
  - **homophones** = two words which sound the same
    - e.g., read/read (past tense), cite/site/sight, they’re/their/there
  - letter/word substitution: replacing a letter (or sequence of letters) with a similar-sounding one
    - e.g., John kracked his nuckles.
      instead of John cracked his knuckles.

**Challenges & Techniques for spelling correction**

Before we turn to how we detect spelling errors, we’ll look briefly at three issues:

- **Tokenization**: What is a word?
- **Inflection**: How are some words related?
- **Productivity of language**: How many words are there?

How we handle these issues determines how we build a dictionary.

And then we’ll turn to the techniques used:

- Non-word error detection
- Isolated-word error correction
- Context-dependent word error detection and correction → grammar correction

**Inflection**

- A word in English may appear in various guises due to word **inflections** = word endings which are fairly systematic for a given part of speech
  - plural noun ending: the boy + s → the boys
  - past tense verb ending: walk + ed → walked
  - This can make spell-checking hard:
    - There are exceptions to the rules: *mans,* *runned*
    - There are words which look like they have a given ending, but they don’t: Hans, deed

**Productivity**

- part of speech change: nouns can be verbified
  - emailed is a common new verb coined after the noun email
- morphological productivity: prefixes and suffixes can be added
  - e.g., I can speak of un-email-able for someone who you can’t reach by email.
- words entering and exiting the lexicon, e.g.:
  - thou, or splet’t split’ (Hamlet I.3.10) are on their way out
  - New words all the time: omnishambles, phablet, supersize, ...

**Grammar correction rules**

- **Minimum edit distance**
  - similarity key techniques
  - rule-based methods
  - N-gram analysis
  - dictionaries

**Knowledge problems**

- not knowing a word and guessing its spelling (can be phonetic)
  - e.g., sientist
- not knowing a rule and guessing it
  - e.g., Do we double a consonant for ing words?
    - jog → jokging
  - knowing something is odd about the spelling, but guessing the wrong thing
    - e.g., typing sicsors for the non-regular scissors
Non-word error detection

And now the techniques ...

- non-word error detection is essentially the same thing as word recognition = splitting up "words" into true words and non-words.
- How is non-word error detection done?
  - using a dictionary (construction and lookup)
  - n-gram analysis

Dictionary construction

- Do we include inflected words? i.e., words with prefixes and suffixes already attached.
  - Lookup can be faster
  - But takes more space & doesn't account for new formations (e.g., google -> googled)
- Want the dictionary to have only the word relevant for the user → domain-specificity
  - e.g., For most people memoize is a misspelled word, but in computer science this is a technical term
- Foreign words, hyphenations, derived words, proper nouns, and new words will always be problems
  - we cannot predict these words until humans have made them words.
- Dictionary should be dialectally consistent.
  - e.g., include only color or colour but not both

Dictionaries

Intuition:

- Have a complete list of words and check the input words against this list.
- If it's not in the dictionary, it's not a word.

Two aspects:

- Dictionary construction = build the dictionary (what do you put in it?)
- Dictionary lookup = lookup a potential word in the dictionary (how do you do this quickly?)

N-gram analysis

- An n-gram here is a string of n letters.
  
  a
  1
  at
  ate
  late
  4

- We can use this n-gram information to define what the possible strings in a language are.
  - e.g., po is a possible English string, whereas kvl is not.
  
  This is more useful to correct optical character recognition (OCR) output, but we'll still take a look.

Bigram array

- We can define a bigram array = information stored in a tabular fashion.
- An example, for the letters k, l, m, with examples in parentheses

<table>
<thead>
<tr>
<th></th>
<th>k</th>
<th>l</th>
<th>m</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>0</td>
<td>1 (tackle)</td>
<td>1 (Hackman)</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>1 (elk)</td>
<td>1 (hello)</td>
<td>1 (alms)</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>0</td>
<td>1 (hammer)</td>
<td></td>
</tr>
</tbody>
</table>

This is the first letter of the bigram is given by the vertical letters (i.e., down the side), the second by the horizontal ones (i.e., across the top).

This is a non-positional bigram array = the array 1's and 0's apply for a string found anywhere within a word (beginning, 4th character, ending, etc.).

Positional bigram array

- To store information specific to the beginning, the end, or some other position in a word, we can use a positional bigram array = the array only applies for a given position in a word.

- Here's the same array as before, but now only applied to word endings:

<table>
<thead>
<tr>
<th></th>
<th>k</th>
<th>l</th>
<th>m</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>1 (elk)</td>
<td>1 (hall)</td>
<td>1 (elm)</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Isolated-word error correction

- Having discussed how errors can be detected, we want to know how to correct these misspelled words:
  - The most common method is isolated-word error correction = correcting words without taking context into account.
  - Note: This technique can only handle errors that result in non-words.
  - Knowledge about what is a typical error helps in finding correct word.

Isolated-word error correction methods

- Many different methods are used; we will briefly look at four methods:
  - rule-based methods
  - similarity key techniques
  - probabilistic methods
  - minimum edit distance
- The methods play a role in one of the three basic steps:
  1. Detection of an error (discussed above)
  2. Generation of candidate corrections
     - rule-based methods
     - similarity key techniques
  3. Ranking of candidate corrections
     - probabilistic methods
     - minimum edit distance

Knowledge about typical errors

- word length effects: most misspellings are within two characters in length of original
  - When searching for the correct spelling, we do not usually need to look at words with greater length differences.
- first-position error effects: the first letter of a word is rarely erroneous
  - When searching for the correct spelling, the process is sped up by being able to look only at words with the same first letter.

Rule-based methods

One can generate correct spellings by writing rules:

- Common misspelling rewritten as correct word:
  - e.g., *hte* → *the*
- Rules
  - based on inflections:
    - e.g., VCing → VCCing, where
      - letter representing vowel,
      - basically the regular expression \[aeiou]\]
      - letter representing consonant,
      - basically \[bcdfghjklmnpqrstvwxyz\]
  - based on other common spelling errors (such as keyboard effects or common transpositions):
    - e.g., CsC → CaC
    - e.g., cie → cei

Similarity key techniques (SOUNDEX)

- Problem: How can we find a list of possible corrections?
- Solution: Store words in different boxes in a way that puts the similar words together.
- Example:
  1. Start by storing words by their first letter (first letter effect),
     - e.g., punc starts with the code P.
  2. Then assign numbers to each letter
     - e.g., 0 for vowels, 1 for b, p, f, v (all bilabials), and so forth, e.g., punc → P052
  3. Then throw out all zeros and repeated letters,
     - e.g., P052 → PS2
  4. Look for real words within the same box,
     - e.g., punk is also in the PS2 box.

How is a mistyped word related to the intended?

For ranking errors, it helps to know:

Types of operations

- insertion = a letter is added to a word
- deletion = a letter is deleted from a word
- substitution = a letter is put in place of another one
- transposition = two adjacent letters are switched

Note that the first two alter the length of the word, whereas the second two maintain the same length.
Confusion probabilities

- It is impossible to fully investigate all possible error causes and how they interact, but we can learn from watching how often people make errors and where.

One way is to build a **confusion matrix** — a table indicating how often one letter is mistyped for another:

<table>
<thead>
<tr>
<th></th>
<th>r</th>
<th>s</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>correct</td>
<td></td>
<td>n/a</td>
<td>12</td>
</tr>
<tr>
<td>typed</td>
<td>s</td>
<td>14</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>t</td>
<td>11</td>
<td>37</td>
</tr>
</tbody>
</table>

(cf. Kernighan et al 1999)

- We also calculate probabilities for insertions & deletions after particular letters.
- Now, let's say...
**Bayes Rule**

With \( X \) as the correct word and \( Y \) as the misspelling ...

\[ P(Y|X) = \text{the probability of the observed misspelling given the correct word} \]

\[ P(X) = \text{the probability of the (correct) word occurring anywhere in the text} \]

Bayes Rule allows us to calculate \( P(X|Y) \) in terms of \( P(Y|X) \):

\[ P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} \]

**Finding the Correct Spelling**

Goal: for a given misspelling \( y \), find correct spelling \( x = \arg \max_x \frac{Pr(y|x)Pr(x)}{Pr(y)} \)

1. List “all” possible candidate corrections, i.e., all words with one insertion, deletion, substitution, or transposition
2. Rank them by their probabilities

Example: calculate for donald

\[ Pr(\text{donald}|\text{donald})Pr(\text{donald}) \]

and see if this value is higher than for any other possible correction.

**Minimum edit distance**

In order to rank possible spelling corrections, it can be useful to calculate the minimum edit distance = minimum number of operations it would take to convert one word into another.

For example, we can take the following five steps to convert junk to haiku:

1. junk → juk (deletion)
2. juk → huk (substitution)
3. huk → hku (transposition)
4. hku → hiku (insertion)
5. hiku → haiku (insertion)

But is this the minimal number of steps needed?

**The Noisy Channel and Bayes Rule**

We can directly relate Bayes Rule to the Noisy Channel:

\[ \frac{P(Y|X)}{P(Y)} \]

Goal: for a given \( y \), find \( x = \arg \max_x \frac{Pr(y|x)Pr(x)}{Pr(y)} \)

The denominator is ignored because it’s the same for all possible corrections, i.e., the observed word \( y \) doesn’t change.

**Obtaining probabilities**

How do we get these probabilities?

We can count up the number of occurrences of \( X \) to get \( P(X) \), but where do we get \( P(Y|X) \)?

- We can use confusion matrices, as we saw before: one matrix each for insertion, deletion, substitution, and transposition
- These matrices are calculated by counting how often, e.g., \( ab \) was typed instead of \( a \) in the case of insertion

To get \( P(Y|X) \), then, we find the probability of this kind of typo in this context.

For insertion, for example (\( X_p \) is the \( p \)th character of \( X \)):

\[ P(Y|X) = \frac{\text{ins}[X_{p-1}, Y]}{\text{count}[X_{p-1}]} \]

**Computing edit distances**

Figuring out the upper bound

- To be able to compute the edit distance of two words at all, we need to ensure there is a finite number of steps.
- This can be accomplished by
  - requiring that letters cannot be changed back and forth a potentially infinite number of times, i.e., we
  - limit the number of changes to the size of the material we are presented with, the two words.
- Idea: Never deal with a character in either word more than once.
- Result:
  - We could delete each character in the first word and then insert each character of the second word.
  - Thus, we will never have a distance greater than \( \text{length(word1)} + \text{length(word2)} \)
To calculate minimum edit distance, we set up a directed, acyclic graph, a set of nodes (circles) and arcs (arrows).

- Horizontal arcs correspond to deletions, vertical arcs correspond to insertions, and diagonal arcs correspond to substitutions (a letter can be “substituted” for itself).

We want to find the path from the start (A) to the end (T) with the least cost.

- The simple but dumb way of doing it:
  - Follow every path from start (A) to finish (T) and see how many changes we have to make.
  - But this is very inefficient! There are many different paths to check.

The key point is that we are storing partial results along the way, instead of recalculating everything, every time we compute a new path.

We compute a new path.

- The smart way to compute the least cost uses dynamic programming = a program designed to make use of results computed earlier
  - We follow the topological ordering.
  - As we go in order, we calculate the least cost for that node:
    - We add the cost of an arc to the cost of reaching the node this arc originates from.
    - We take the minimum of the costs calculated for all arcs pointing to a node and store it for that node.

The smart way to compute the least cost uses dynamic programming = a program designed to make use of results computed earlier

To calculate minimum edit distance, we set up a directed, acyclic graph, a set of nodes (circles) and arcs (arrows).

- Horizontal arcs correspond to deletions, vertical arcs correspond to insertions, and diagonal arcs correspond to substitutions (a letter can be “substituted” for itself).

The key point is that we are storing partial results along the way, instead of recalculating everything, every time we compute a new path.

The smart way to compute the least cost uses dynamic programming = a program designed to make use of results computed earlier

- We follow the topological ordering.
  - As we go in order, we calculate the least cost for that node:
    - We add the cost of an arc to the cost of reaching the node this arc originates from.
    - We take the minimum of the costs calculated for all arcs pointing to a node and store it for that node.

The key point is that we are storing partial results along the way, instead of recalculating everything, every time we compute a new path.

The smart way to compute the least cost uses dynamic programming = a program designed to make use of results computed earlier

- We follow the topological ordering.
  - As we go in order, we calculate the least cost for that node:
    - We add the cost of an arc to the cost of reaching the node this arc originates from.
    - We take the minimum of the costs calculated for all arcs pointing to a node and store it for that node.

The key point is that we are storing partial results along the way, instead of recalculating everything, every time we compute a new path.
Spelling correction for web queries is hard because it must handle:
- Proper names, new terms, etc. *(blog, shrek, nsync)*
- Frequent and severe spelling errors
- Very short contexts

**Algorithm**

Main Idea *(Cucerzan and Brill (EMNLP-04))*

- Iteratively transform the query into more likely queries
- Use query logs to determine likelihood
- Despite the fact that many of these are misspelled!
- Assumptions: the less wrong a misspelling is, the more frequent it is; and correct > incorrect

**Example:**

```
anol scwartegger
anol schwartnegger
anol schwarznegger
anol schwarzenegger
```

**Algorithm (2)**

- Compute the set of all close alternatives for each word in the query
  - Look at word unigrams and bigrams from the logs; this handles concatenation and splitting of words
  - Use weighted edit distance to determine closeness
  - Search sequence of alternatives for best alternative string, using a noisy channel model

**Constraint:**
- No two adjacent in-vocabulary words can change simultaneously

**The formal algorithm** *(just for fun)*

Given a string $s_0$, find a sequence $s_1, s_2, \ldots, s_n$ such that:
- $s_n = s_{n-1}$ (stopping criterion)
- $\forall i \in 0 \ldots n-1,$
  - $\text{dist}(s_i, s_{i+1}) \leq \delta$ (only a minimal change)
  - $P(s_{i+1}|s_i) = \max_t P(t|s_i)$ (the best change)

**Examples**

Context Sensitivity

- power crd \rightarrow power cord
- video crd \rightarrow video card
- platinum rings \rightarrow platinum rings

Known Words

- golf war \rightarrow gulf war
- sap opera \rightarrow soap opera

**Examples (2)**

Tokenization

- chat inspanich \rightarrow chat in spanish
- ditrogliders \rightarrow detroit tigers
- britenetspear inconcert \rightarrow britney spears in concert

Constraints

- log wood \rightarrow log wood (not dog food)
Context-dependent word correction

**Context-dependent word correction** = correcting words based on the surrounding context.

- This will handle errors which are real words, just not the right one or not in the right form.
- This is very similar to a **grammar checker** = a mechanism which tells a user if their grammar is wrong.

More on grammar correction

- Semantic errors = errors where the sentence structure sounds okay, but it doesn’t really mean anything.
  - e.g., They are leaving in about fifteen minuets to go to her house.
  
  \[\text{minuets and minutes}\] are both plural nouns, but only one makes sense here

There are many different ways in which grammar correctors work, two of which we’ll focus on:

- **N-gram model**
- **Rule-based model**

Rule-based grammar correctors

We can write regular expressions to target specific error patterns. For example:

- **To a certain extend, we have achieved our goal.**
  - Match the pattern some or certain followed by extend, which can be done using the regular expression some\(\text{certain}\)\text{extend}
  - Change the occurrence of \text{extend} in the pattern to extent.


Beyond regular expressions

- But what about correcting the following:
  - A baseball **teams** were successful.
  - We should see that A is incorrect, but a simple regular expression doesn’t work because we don’t know where the word teams might show up.
  - \(\text{A wildly overpaid, horrendous baseball teams were successful.}\) (Five words later; change needed.)
  - **A player on both my teams was successful.** (Five words later; no change needed.)

We need to look at how the sentence is constructed in order to build a better rule.

Grammar correction—what does it correct?

- **Syntactic errors** = errors in how words are put together in a sentence: the order or form of words is incorrect, i.e., ungrammatical.
  - **Local syntactic errors:** 1-2 words away
    - e.g., \(\text{The study was conducted mainly is John Black.}\)
    - A verb is where a preposition should be.
  - **Long-distance syntactic errors:** (roughly) 3 or more words away
    - e.g., \(\text{The kids who are most upset by the little totem is going home early.}\)
    - Agreement error between subject kids and verb is

- **Semantic errors** = errors where the sentence structure is incorrect, but a simple regular expression can be done using the regular expression

\[\text{bigrams}\] of words, i.e., two words appearing next to each other.

- **Question:** Given the previous word, what is the probability of the current word?
  - e.g., given these, we have a lower chance of seeing \text{reports} than of seeing \text{report}.
  - Since a confusable word (\text{reports}) can be put in the same context, resulting in a higher probability, we flag \text{report} as a potential error
  
  But there’s a major problem: we may hardly ever see \text{these reports}, so we won’t know its probability.

- **Possible Solutions:**
  - use bigrams of parts of speech
  - use massive amounts of data and only flag errors when you have enough data to back it up
**Syntax**

- **Syntax** = the study of the way that sentences are constructed from smaller units.
- There cannot be a “dictionary” for sentences since there is an infinite number of possible sentences:
  1. The house is large.
  2. John believes that the house is large.
  3. Mary says that John believes that the house is large.

There are two basic principles of sentence organization:

- **Linear order**
- **Hierarchical structure (Constituency)**

**Constituency**

- What are the “meaningful units” of a sentence like *Most of the ducks play extremely fun games*?
  1. Most of the ducks
  2. of the ducks
  3. extremely fun
  4. extremely fun games
  5. play extremely fun games
- We refer to these meaningful groupings as **constituents** of a sentence.

**Lexical categories** are simply word classes, or what you may have heard as **parts of speech**. The main ones are:

- verbs: *eat, drink, sleep, ...*
- nouns: *gas, food, lodging, ...*
- adjectives: *quick, happy, brown, ...*
- adverbs: *quickly, happily, well, westward*
- prepositions: *on, in, at, to, into, of, ...*
- determiners/articles: *a, an, the, this, these, some, much, ...*

---

**Linear order**

- **Linear order** = the order of words in a sentence.
- A sentence can have different meanings, based on its linear order:
  1. (6) John loves Mary.
  2. (7) Mary loves John.
- Languages vary as to what extent this is true, but linear order in general is used as a guiding principle for organizing words into meaningful sentences.
- Simple linear order as such is not sufficient to determine sentence organization though. For example, we can’t simply say “The verb is the second word in the sentence.”
  1. (8) **I eat** at really fancy restaurants.
  2. (9) Many executives **eat** at really fancy restaurants.
Determining lexical categories

How do we determine which category a word belongs to?
- **Distribution**: Where can these kinds of words appear in a sentence?
  - e.g., Nouns like mouse can appear after articles ("determiners") like some, while a verb like eat cannot.
- **Morphology**: What kinds of word prefixes/suffixes can a word take?
  - e.g., Verbs like walk can take a ed ending to mark them as past tense. A noun like mouse cannot.

Building a tree

Other phrases work similarly (S = sentence, VP = verb phrase, PP = prepositional phrase, AdjP = adjective phrase):

```
  S
    NP
      Pro
        PP
          Most
            P
              N
                of
                  D
                    Adv
                      Adj
                        games
                    Adv
                      Adj
                        N
                          Adv
                            Adj
                              N
                                Adv
                                  Adj
                                    N
                                      Adv
                                        Adj
                                          N
                                            Adv
                                              Adj
                                                N
                                                  Adv
                                                    Adj
                                                      N
                                                        Adv
                                                          Adj
                                                            N
                                                              Adv
                                                                Adj
                                                                  N
                                                                    Adv
                                                                      Adj
                                                                        N
```

Phrase Structure Rules

- We can give rules for building these phrases. That is, we want a way to say that a determiner and a noun make up a noun phrase, but a verb and an adverb do not.
- **Phrase structure rules** are a way to build larger constituents from smaller ones.
  - e.g., $S \rightarrow NP \ VP$

  This says:
  - A sentence (S) constituent is composed of a noun phrase (NP) constituent and a verb phrase (VP) constituent. [Hierarchy]
  - The NP must precede the VP. [linear order]

Some other possible English rules

- $NP \rightarrow Det\ N$ (the cat, a house, this computer)
- $NP \rightarrow Det\ AdjP\ N$ (the happy cat, a really happy house)
  - For phrase structure rules, as shorthand parentheses are used to express that a category is optional.
  - We thus can compactly express the two rules above as one rule:
    - $NP \rightarrow Det(\ AdjP)\ N$
  - Note that this is different and has nothing to do with the use of parentheses in regular expressions.
  - $AdjP \rightarrow (Adv)\ Adj$ (really happy)
  - $VP \rightarrow V$ (laugh, run, eat)
  - $VP \rightarrow V\ NP$ (love John, hit the wall, eat cake)
  - $VP \rightarrow V\ NP\ NP$ (give John the ball)
  - $PP \rightarrow P\ NP$ (to the store, at John, in a New York minute)
  - $NP \rightarrow NP\ PP$ (the cat on the stairs)

Phrase Structure Rules and Trees

With every phrase structure rule, you can draw a tree for it.

**Lexicon:**
- $Vt \rightarrow saw$
- $Det \rightarrow the$
- $Det \rightarrow a$
- $N \rightarrow dragon$
- $N \rightarrow boy$
- $Adj \rightarrow young$

**Syntactic rules:**
- $S \rightarrow NP\ VP$
- $VP \rightarrow Vt\ NP$
- $NP \rightarrow Det\ N$
- $N \rightarrow AdjN$
Properties of Phrase Structure Rules

- **generative** = a schematic strategy that describes a set of sentences completely.
- potentially **structurally ambiguous** = have more than one analysis

(10) We need more intelligent leaders.

(11) Paraphrases:
   a. We need leaders who are more intelligent.
   b. Intelligent leaders? We need more of them!

- **recursive** = property allowing for a rule to be reapplied (within its hierarchical structure).
  
  e.g., NP → NP PP
  
  PP → P NP

- The property of recursion means that the set of potential sentences in a language is infinite.

Context-free grammars

A context-free grammar (CFG) is essentially a collection of phrase structure rules.

- It specifies that each rule must have:
  - a left-hand side (LHS): a single non-terminal element = (phrasal and lexical categories)
  - a right-hand side (RHS): a mixture of non-terminal and terminal elements = actual words

- A CFG tries to capture a natural language completely.

Why “context-free”? Because these rules make no reference to any context surrounding them. i.e. you can’t say “PP → P NP” when there is a verb phrase (VP) to the left.

Pushdown automata

**Pushdown automaton** = the computational implementation of a context-free grammar.

It uses a stack (its memory device) and has two operations:

- **push** = put an element onto the top of a stack,
- **pop** = take the topmost element from the stack.

This has the property of being **Last In First Out (LIFO)**.

Consider a rule like PP → P NP

- Push NP onto the stack
- Push P onto it
- If you find a preposition (e.g., on), pop P off of the stack

  ▶ Now, the next thing you need is an NP ... when you find that, pop NP and push PP onto the stack.

Parsing

Using these context-free rules and something like a pushdown automaton, we can get a computer to parse a sentence = assign a structure to a sentence.

There are many, many parsing techniques out there.

- **top-down**: build a tree by starting at the top (i.e. S → NP VP) and working down the tree.
- **bottom-up**: build a tree by starting with the words at the bottom and working up to the top.

Trace of a top-down parse

Trace of a bottom-up parse
Writing grammar correction rules

So, with context-free grammars, we can now write some correction rules, which we will just sketch here.

- A baseball team was successful.
  - A followed by PLURAL NP: change A → The
- John at the pizza.
  - The structure of this sentence is NP PP, but that doesn’t make up a whole sentence.
  - We need a verb somewhere.

A Poem on the Dangers of Spell Checkers

Michael Livingston

Eye have a spelling chequer
It plainly marques four my revue
Miss steaks eye kin knot sea.
Eye strike a key and type a word
And weight four it two say
Weather eye am wrong oar write
It shows me strait a weigh.
As soon as a mist ache is maid
It nose bee fore two long
And eye can put the error rite
Its rare lea ever wrong.
Eye have run this poem threw it
I am shore your pleased two no
Its letter perfect awl the weigh
My chequer tolled me sew.

Dangers of spelling and grammar correction

- The more we depend on spell correctors, the less we try to correct things on our own. But spell checkers are not 100%
- A study at the University of Pittsburgh found that students made more errors (in proofreading) when using a spell checker!

<table>
<thead>
<tr>
<th>use checker</th>
<th>high SAT scores</th>
<th>low SAT scores</th>
</tr>
</thead>
<tbody>
<tr>
<td>no checker</td>
<td>16 errors</td>
<td>17 errors</td>
</tr>
<tr>
<td></td>
<td>5 errors</td>
<td>12.3 errors</td>
</tr>
</tbody>
</table>

(cf., http://www.wired.com/news/business/0,1367,58058,00.html)

References